Building a database from scratch
benoît chesneau

craftsman working on *P2P and custom data endpoints technologies*

opensource only

enki multimedia : the corporate interface
- versatile data endpoint
- micro-services, message solutions are all based about custom data endpoints
- need for a simple solution that allows you to bring the data near your service or locally.
a modern **database**

**documents**, with time and attachments

distributed, **local first**

bring a **view of your data near your application**

**automatic indexing**

focus on simplicity

what is **barrel**?
a partial view of the data
a partial view of the data
agnostic indexing

```json
{ "locations": [  
    { "country": "Germany", "city": "Berlin" },
    { "country": "France", "city": "Paris" }
],
"headquarter": "Belgium",
"exports": [{ "city": "Moscow" },
    { "city": "Athens"}],
}

{ "locations": [  
    { "country": "Germany",  
      "city": "Bonn","revenue": 200
    },
    { "country": "Italy",  
      "headquarter": "Italy",  
      "exports": [  
        { "city": "Berlin","dealers": [{"name": "Hans"}]},
        { "city": "Athens" }
    ]
},
```

![Diagram of Document 1](image1.png)

![Diagram of Document 2](image2.png)
barrel can be embedded in your own Erlang application:

- local database
- no need to cache
- platform release: HTTP/Erlang pod to store and query the documents
problems to solve
- stateful
- different queries return different results
- update expectations
  - read your own write?
- processes don’t share anything
  - how do we have multiple writers and multiple readers
  - actor model
- no integer atomic operations
- IO operations are “slow”
  - until you get nifs
- build over existing storage solutions:
  - key/value interface
  - allows atomic batch updates
  - ordered set

- 1 collection, 1 storage

- collections are small
multiple collections on a node

a collection

 disobedience / db / docs

store

hierarchical
- document:
  - map in erlang

- revision tree:
Body of leaf node in winning rev tree must be kept

Body of leaf node in losing rev trees must be kept
2 modes: lazy and consistent

- lazy: indexed asynchronously based on the changes feed
- consistent

- support maps, filter, chain operations based on paths
internals
- using rocksdb for the storage
  - http://gitlab.com/barrel-db/erlang-rocksdb
- used for memory and disk. optimised for SSD.
- dirty nifs
- writes are queued on the main db process
- store a canonical version of doc
- states of the database is shared between other processes via ETS
  - readers are getting last db state via ets

write process (current)
prevent delayed jobs
- write more operations at once
  - selective receive
- group operations based on the document ID (merge)
- from 40 RPS to 1000 RPS on a node with 4GB of ram and 2 cores

write process (current)
- By ID, Changes queries
- get latest DB state from ETS
- everything happen on the reader process
- coming: backpressure
  - share the db state across a pool of readers
  - remove the state from ETS
- testing dispatching of write operations on different processes:

- testing optimistic writes

- back pressure:
  - short circuit to not accept more write than the node can sustain
  - based on the running transaction and metrics

- similar to safety valve:
  - [https://github.com/jlouis/safetyvalve](https://github.com/jlouis/safetyvalve)
- just appending data to the storage we never read from old index values
- inside the DB process for consistent write
- a process listening on db updates events (using a simple gen_server, no gen_event)
- index policies to index each json segment to retrieve via their value or hash to support value or range queries.
- over HTTP
  - cowboy 2
- over TCP using teleport and Erlang serialisation (coming):
  - https://gitlab.com/barrel-db/teleport
- allows embedded mode
add some instrumentation
- how to not block without counting
- first try: statsd client sending to an UDP endpoint counter/gauge/histogram updates
- we run out of processes & file descriptors
- asynchronous sending: better.
- how to make generic?
- add hooks
  - https://github.com/benoitc/hooks
- prometheus plugin and wombat support (EE version)
- internal metrics system
  - https://gitlab.com/barrel-db/lab/instrument

```erlang
barrel_start_transaction(Trans, DbName) ->
erlang:put(barrel_transaction_start_time, erlang:monotonic_time()),
prometheus_counter:inc(barrel_db_transactions, [DbName, Trans]).
```
roamap
- 0.9 release: 2017/06/13
  - https://gitlab.com/barrel-db/barrel-platform
- add documentation (June 2017)
- optimise writing
- atomic updates
- enrich query engine.

roadmap